DFA for Efficient Regular Expressions Matching

Mithilesh Kumar Singh Yadav [1], Dr. Ajit Kumar Singh [2]
Assistant Professor [1], Professor [2] Department of Computer Science and Engineering
[1] [2] Sherwood College of Engg. Research & Technology, Luck now
Email: [mithilesh.mnnit, ajit2504singh]@gmail.com

Abstract—Most of the network security applications in today's networks are based on Deep Packet Inspection (DPI), is a form of computer network packet filtering that examines not only the header portion but also the payload part of a packet as it passes an inspection point, searching for protocol noncompliance, viruses, spam, intrusions or predefined criteria to decide if the packet can pass or if it needs to be routed to a different destination, or for the purpose of collecting statistical information. Most high performance systems that perform deep packet inspection implement simple string matching algorithms to match packets against large (finite) strings. Network intrusion de-tection systems (NIDS) are among the most widely deployed such system. NIDS uses a collection of signatures of known security threats and viruses, which are used to scan each packets payload. However, there is growing interest in the use of regular expression-based pattern matching, since regular expressions offer superior expressive power. However, DFA representations of regular expression sets arising in network applications require large amounts of memory, limiting their practical application. This paper presents a new representation for DFA called default transition finite automata (dFA), which considerably reduces the number of transition by replacing several transitions by a default transition.

KeyWords: **Network intrusion detection systems, Regular expressions, DFA.**

I. INTRODUCTION

Many network security applications in today's networks are based on deep packet inspection, checking not only the header portion but also the payload portion of a packet. Traffic monitoring, network intrusion detection all require an accurate analysis of packet content in search for predefined patterns to identify specific classes of applications, viruses, attack signatures, etc. Those patterns were traditionally a number of strings representing signatures to be compared against packet contents using exact matching algorithms. However, exact matching is not expressive enough to detect malicious patterns with the evolution of the network threats. Thus, more expressive regular expressions are used to describe a wide variety of signatures. For example, Snort, an open-source network intru-sion detection system, also uses regular expressions as its signature language [9]. Bro, another open-source intrusion detection system, also uses regular expressions as its signature language [8].

These regular expressions are also used in commercial firewalls and other networking equipment. The most popular method to implement regular expression matching is to use finite automata. The finite automaton is either deterministic or non-deterministic. A non-deterministic finite automaton (NFA) may have many state transitions per character. However, it is very efficient in terms of space usage compared to a deterministic counterpart. On the other hand, a deterministic finite automaton (DFA) has only one state transition per character, while it requires a much larger amount of memory for the same regular expression [1]. Therefore, DFAs are more suitable for general-purpose processors and network processors. However, DFA representations of regular expression sets arising in network applications require large amounts of memory, limiting their practical application.

This paper, introduces a novel compact representation scheme, default transition finite automata (named dFA), which considerably reduces the number of transition by replacing several transitions by a default transition. Reducing the redundancy of transitions appears to be very appealing since the recent general trend in the proposals for compact and fast DFAs construction suggests that the information should be moved toward edges rather than states.

The dFA (default transition FA), introduce a new

representation for regular expressions, which substantially reduces space requirements as compared to a DFA. A dFA is constructed by transforming a DFA via incrementally replacing several transitions of the automaton with a single default transition. This approach dramatically reduces the number of distinct transitions between states.

Rest of the paper is organized as follow: In section II we have described the background of this paper.In section III, we explains the working model with Motivational examples and In section IV we are converting the DFAs to dFAs and describing about the Lemma.

II. BACKGROUND

Network intrusion detection systems (NIDS) are the most widely deployed system. NIDS uses a collection of signatures of known security threats and viruses, which are used to scan each packets payload. However, there is growing interest in the use of regular expression-based pattern matching, since regular expressions offer superior expressive power. These regular expressions are also used in commercial firewalls and other networking equipment. The most popular method to implement regular expression matching is to use finite automata. The finite automaton is either deterministic or non-deterministic. A non-deterministic finite automaton (NFA) may have many state transitions per character. However, it is very efficient in terms of space usage compared to a deterministic counterpart. On the other hand, a deterministic finite automaton (DFA) has only one state transition per character, while it requires a much larger amount of memory for the same regular expression. Therefore, DFAs are more suitable for general-purpose processors and network processors. However, DFA representations of regular expression sets arising in network applications require large amounts of memory, limiting their practical application. While DFAs have been more popular, the merits of NFA-based approaches are increasingly appreciated [5][6]. Beyond the choice of automata, there are three basic algorithmic techniques used to create feasible automata for high-speed regular expression evaluation: (i) edge compression, (ii) alphabet-reduction, and (iii) increased stride[10].

Edge compression reduces the size of a state-minimized DFA by exploiting the redundancy present in the transitions between states. Such is the case with dFA [2][3], where default paths are constructed to eliminate redundant edges.

Alphabet reduction is a basic technique for mapping the set of symbols found in an alphabet to a smaller set by grouping characters that label the same transitions everywhere in the automaton [3][4]. A reduced alphabet size can dramatically diminish the amount of storage needed to represent transitions within an automaton.

Multi-stride DFAs were proposed in [4] as a way to increase processing throughout. Specifically, a stride-k DFA consumes k characters per state transition rather than just one, thus yielding a k-fold performance increase.

III. WORKING MODEL OF d FA

It is well-known that for any regular expression set, there exists a DFA with the minimum number of states. The memory needed to represent a DFA is determined by the number of transitions from one state to another, or equivalently, the number of edges in the graph representation. For an ASCII alphabet, there can be up to 256 edges leaving each state, making the space requirements excessive. The dFA (default transition FA), introduce a new representation for regular expressions, which substantially reduces space requirements as compared to a DFA. A dFA is constructed by transforming a DFA via incrementally replacing several transitions of the automaton with a single default transition. This approach dramatically reduces the number of distinct transitions between states.

Motivational Example

In this DFA, state 1 is the initial state, and states 2, 5 and 4 are match states form the three patterns, p1 = 1^{+}, p2 = 2^{+3}, and p3 = $3\,4^{+}$ (in these expressions, the asterisk represents 0 or more repetitions of the immediately preceding

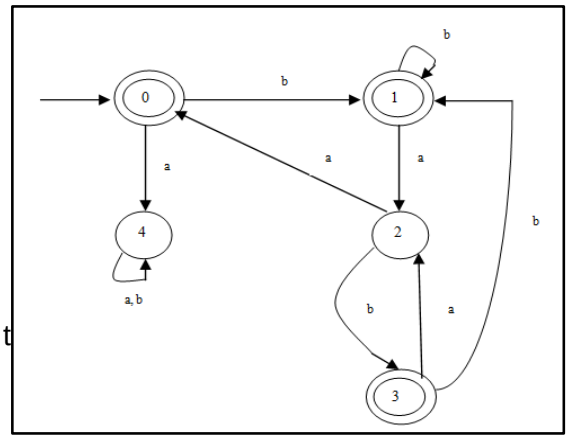

Figure.1

.

Lemma

Consider a dFA with distinct states A and B, where A has a transition labeled by the symbol 1, and no outgoing default transition. If (1; A)= (1; B), then the dFA obtained by introducing a default transition from A to B and removing the transition from u to (1; A) is equivalent to the original DFA.

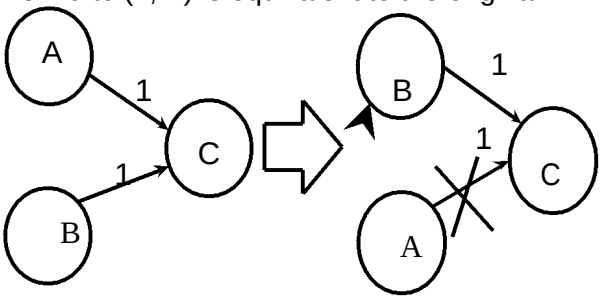

Figure.2

IV. CONVERTING DFAS TO DFAS

Although, we are in general interested in any equivalent dFA, for a given DFA, we have no general procedure for synthesizing a dFA directly. Consequently, our procedure for constructing a dFA proceeds by transforming an ordinary DFA, by introducing default transitions in a systematic way, while maintaining equivalence. Our procedure does not change the state set, or the set of matching patterns for a given state. Hence, we can maintain equivalence by ensuring that the destination state function (x), does not change[7].

Consider two states A and B, where both A and B have a transition labeled by the symbol a to a common third state C, and no default transition (unlabeled outgoing transition). If we introduce a default transition from A to B, we can eliminate the a-transition from A without affecting the destination state function (x)[7]. A slightly more general version of this observation is stated below Regular expression dataset R is input to the first phase. The Regular expression illustrates the pattern of the string. These regular expressions contains characters with symbols used in regular expressions such as closure (*) for zero and more occurrences, or (+) for one and more occurrences etc. Each regular expression r in R is converted into NFA using Thomson algorithm [11]. The set N of NFAs of given regular expressions is an input to the second phase.

Note that by the same reasoning, if there are multiple symbols a, for which A has a labeled outgoing edge and for which (1; A)= B(1; B), the introduction of a default edge from A to allows us to eliminate all these edges. Our procedure for converting a DFA to a smaller dFA applies this transformation repeatedly. Hence, the equivalence of the initial and final dFA s follows by induction. The dFA on the right side of Figure 1 was obtained from the DFA on the left, by applying this transformation to state pairs (2,1), (3,1), (5,1) and (4,1).

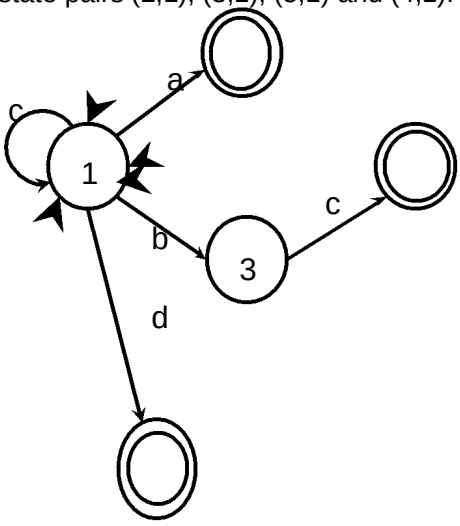

Figure.3

Note that the automaton in Figure 3 has just nine edges, while in Figure 1 has twenty edges.

DFA	Number of Transitions
Figure.1	18
Figure.2	7

Table-1 shows Number of Transition of the original DFA with dFA

V. CONCLUSION

In this paper, we have presented a new compressed representation for deterministic finite automata, called default transition Finite Automata (dFA). The dFA (default transition FA), introduce a new representation for regular expressions, which substantially reduces space requirements as compared to a DFA. A dFA is constructed by transforming a DFA via incrementally replacing several transitions of the automaton with a single default transition. The algorithm considerably reduces the number of transitions.

In a DFA, most adjacent states share several common transitions, so the relation between the adjacent states and the concept of default transition can be taken for reducing the number of transitions. Regular expressions are broadly used to represent signatures of security attacks. DFA is easy way to express regular expressions. Memory space required to store DFA is very large. To address this problem, this paper has described the method which reduced the size of DFA generated from regular expression. The regular expression matching by compressing DFA method has converted regular expressionsinto DFA of minimum size. The DFA is stored into memoryin the form of compressed rules. The compressed DFA of regular expressions is used at the end in regular expression matching process. As a future work, one may consider the regular expression which represents security attacks in special symbols for building and compressing deterministic finite automata.

REFERENCES

[1].J. Hopcroft and J Ullman, “Introduction to Automata Theory, Languages, and Computation”,Addison Wesley, 1979.

[2].S. Kumar et al,“Algorithms to Accelerate Multiple Regular Expressions Matching for Deep Packet Inspection”,In ACM SIGCOMM, Sept 2006.

[3] M. Becchi, P. Crowley, “Efficient Regular Expression Evaluation : Theory to Practice”Theory to Practice, In Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, November 2008, pp 50-59

.

4.M. Becchi and P. Crowley “A Hybrid Finite Automaton for Practical Deep Packet Inspection”. In CoNEXT 2007

[5] Jo n . Hopcroft and Jeffrey D. Ullman, “Introduction to Automata eory, Lan ua es, and Computation”,Addison-Wesley Publishing, Reading Massachusetts

[6] “SNOR ,” 20 0 Online . A ailable: http://www.snort.org

[7] D.Ficara, S.Giordano, G. Procissi, F.Vitucci, G.Antichi, A.D. Pietro, “An Improved DFA for Fast Regular xpression Matc in ” ACM SIGCOMM Computer Communication Review, Volume 38, Number 5, October 2008.

[8] S. Kumar, J. urner, J. Williams, “Ad anced al orit ms for fast and scalable deep packet inspection”, in Proc.ACM/IEEE Symp. Archit. Netw. Commun. Syst.(ANCS), pages 81-92. ACM, 2006.

[9] M. Becc i, P. Crowley, “A ybrid finite automaton for practical deep packet inspection”, in Proc. Conf. Emerging Netw. Exp. Technol.(CoNEXT), pages 1-12, 2007.

[10] S. Kumar, B. Chandrasekaran, J. Turner, G. Varghese,“Curin re ular expressions matc in al orit ms frominsomnia, amnesia, and acalculia”, in Proc. ACM/ISymp. Archit. Netw. Commun. Syst. (ANCS), pages155-164. ACM, 2007

[11] AnatBremler-Barr, D.Hay, Y. Koral,“CompactDFA:Scalable pattern matc in Usin Lon est Prefix Match Solutions," in IEEE/ACM Transaction on networking,vol-22,No.2,April 2014.

[12] A.V. A o and M.J. Corasick. “ fficient Strin Matc in :An Aid to Biblio rap ic Searc .” Communications of t eACM, 18(6):333–340, 1975.

[13] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. urner, “Al orit ms to accelerate multiple regular expressions matc in for deep packet inspection”, in Proc. of ACM SIGCOMM , pages 339-350. ACM, 2006.